15-451/651 Algorithms, Fall 2017

Homework #1 **Due: September 8, 2017**

This HW has three regular problems, and one programming problem. All problems on written HWs are to be done *individually*, no collaboration is allowed.

Solutions to the three written problems should be submitted as a single PDF file using `gradescope`, with the answer to each problem starting on a new page.

Submission instructions for the programming problem will be posted on the website and Piazza.

(25 pts) 1. **Sorting with Strange Primitives**

In both of the parts below an array A initially contains a permutation of $1, \ldots n$. The goal in each case is to give an algorithm to sort the array by applying a linear number of the given primitives.

(a) A *prefix reversal* of A just reverses some prefix of the array. For example if $A = [1, 5, 3, 2, 4]$ this can be turned into $[3, 5, 1, 2, 4]$ with one prefix reversal (of the first 3 elements of A). Give an algorithm for sorting any initial permutation in at most $c_1 n$ prefix reversals. What is c_1 for your algorithm? (Try to make it as small as you can.)

(b) A *block swap* takes a pair of neighboring blocks of A of the same size and exchanges them. For example if $A = [8, 1, 3, 2, 4, 6, 5, 7]$ in one block swap we could get $[4, 6, 5, 7, 8, 1, 3, 2]$, or $[8, 3, 1, 2, 4, 6, 5, 7]$, or $[2, 4, 6, 8, 1, 3, 5, 7]$, but not $[7, 1, 3, 2, 4, 6, 5, 8]$. Give an algorithm for sorting any initial permutation in at most $c_2 n$ block swaps. What is c_2 for your algorithm? (Try to make it as small as you can.)

(25 pts) 2. **Solving Recurrences**

Give a tight asymptotic bound for the following recurrences. In each case explain the technique you use and why your answer is correct.

(a) $T(n) = 2T(\frac{n}{2}) + 1$ for $n > 1$, $\quad T(n) = 1$ for $n \leq 1$

(b) $T(n) = 3T(\frac{n}{2}) + n^3$ for $n > 1$, $\quad T(n) = 1$ for $n \leq 1$

(c) $T(n) = T(\sqrt{n}) + 1$ for $n > 2$, $\quad T(n) = 1$ for $n \leq 2$

(25 pts) 3. **A Diabolical Sorting Algorithm**

Consider this algorithm, devised by one of your professors, for sorting an array of n (a power of two) elements[1]:

```
CRUEL(A[1..n]):
  if n > 1
      CRUEL(A[1..n/2])
      CRUEL(A[n/2+1..n])
      UNUSUAL(A[1..n])
```

```
UNUSUAL(A[1..n]):
  if n = 2
      if A[1] > A[2]                          ⟨⟨the only comparison!⟩⟩
          swap A[1] ↔ A[2]
  else
      for i ← 1 to n/4                        ⟨⟨swap 2nd and 3rd quarters⟩⟩
          swap A[i + n/4] ↔ A[i + n/2]
      UNUSUAL(A[1..n/2])                      ⟨⟨recurse on left half⟩⟩
      UNUSUAL(A[n/2+1..n])                    ⟨⟨recurse on right half⟩⟩
      UNUSUAL(A[n/4+1..3n/4])                 ⟨⟨recurse on middle half⟩⟩
```

(a) Prove that CRUEL sorts any input array. Hints: The first thing you need to do is to figure out what UNUSUAL does, as well as the precondition that holds for the array being given to it. Then the proof will be by induction. When analyzing UNUSUAL think about what happens to the elements that are in the bottom quartile among those being fed into UNUSUAL.

(b) Write a recurrence for the running time of UNUSUAL and solve it.

(c) With that in hand, now write a recurrence for the running time of CRUEL, and solve it.

Interesting side note: The comparisions done by this algorithm do not depend on data values in the array. It's an oblivious algorithm. It could be used to construct a sorting network.

[1] The notation for passing an array as a parameter used in this pseudo-code is just a short-hand for passing the starting and ending indices of a sub-array of the initial one being sorted. The same array is being used at all points and no copying is going on.

(25 pts) 4. **Programming: Sorting with Swaps**

Write a program which takes as input a permutation of the numbers $0, 1, \ldots, n-1$. View these numbers as the contents of a zero-indexed array a. Your program will output a sequence of pairs of indices into a such that swapping those pairs of indices in that order results in a being sorted. Your program should output a shortest possible such sequence (which will be at most $n-1$ in length).

Your algorithm should also have $O(n)$ traditional running time. The running time bound will be 5 seconds. *Please include a comment at the start of your program explaining your algorithm, and why it runs in linear time.*

Details on how to submit, grading policy, etc., will be on the course website and piazza soon.

INPUT: The first line contains n, which is at most 10^6. The second line consists of the numbers $a_0, a_1, \ldots, a_{n-1}$ separated by blanks. These numbers will be a permutation of $0, 1, \ldots, n-1$.

OUTPUT: The first line of output is k, the minimum number of swaps required to sort the array. The following k lines each contain a pair of indices to swap.

For example, if the input is:

```
5
0 1 4 3 2
```

then the output could be

```
1
2 4
```

Or if the input is:

```
6
1 3 2 4 0 5
```

then the output could be:

```
3
3 1
1 4
0 1
```